МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»
ЗАДАНИЕ № 1
Вариант 1 – 2, 2 – 1.6

ОТЧЁТ
о выполненном задании

студента 205 учебной группы факультета ВМК МГУ
Лабутина Антона Александровича

гор. Москва
2019 г.

Постановка задачи

Задача состояла из двух подвариантов.

В подварианте 1 практической работы был дан набор из нескольких систем уравнений Ax=f порядка n×n с невырожденной матрицей A. Нужно было написать программу, решающую систему линейных алгебраических уравнений заданного пользователем размера (n – параметр программы) методом Гаусса и методом Гаусса с выбором главного элемента. Также нужно предусмотреть возможность задания элементов матрицы системы и её правой части как во входном файле данных, так и путем задания специальных формул.

В подварианте 2 практической работы был дан тот же набор из нескольких систем уравнений. Требовалось написать программу их численного решения, использующую алгоритм итерационного метода верхней релаксации. Также требовалось предусмотреть возможность задания элементов матрицы системы и ее правой части как во входном файле данных, так и путём задания специальных формул.

Цели и задачи практической работы

При выполнении практической работы по численным методам были поставлены следующие задачи:

1. Подвариант 1
   1.1. Решить СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента;
   1.2. Вычислить определитель матрицы;
   1.3. Вычислить обратную матрицу;
   1.4. Исследовать вопрос вычислительной устойчивости метода Гаусса;
2. Подвариант 2
   2.1. Решить заданную СЛАУ итерационным методом верхней релаксации;
   2.2. Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заданной точностью;
   2.3. Изучить скорость сходимости итераций к точному решению задачи в зависимости от итерационного параметра ω;

Эти задачи реализованы в виде программы, прикреплённой к отчёту.

Алгоритмы решения

Решение СЛАУ методом Гаусса:

Равносильными преобразованиями строк матрицы коэффициентов матрица приводится к верхнему треугольному виду (прямой ход). Далее происходит последовательное вычисление корней уравнений, начиная с последней строки матрицы (обратный ход).

Решение СЛАУ методом Гаусса с выбором главного элемента:

В данном методе ведущим элементом выбирается максимальный по модулю элемент строки. Далее алгоритм повторяет алгоритм обычного метода Гаусса.

Вычисление определителя матрицы:

Определитель находился как произведение диагональных элементов матрицы, заранее приведённой к верхней треугольной форме методом Гаусса, с учётом знаком, определяемого чётностью числа перестановок строк.

Нахождение обратной матрицы:

Вычисление обратной матрицы реализовано методом Гаусса, то есть данная матрица приводится равносильными преобразованиями строк к единичной матрице, в то время как к единичной матрице применяются те же преобразования строк; в результате на её месте окажется матрица, обратная к данной.

Решение СЛАУ методом верхней релаксации:

Данный метод вычисляет приближенные решения СЛАУ по формуле:

$$\left(D + \omega A^{(-)}\right)\frac{x^{k+1} - x^k}{\omega} + A x^k = f,$$

где D, $A^{(-)}$, – соответственно диагональная и нижняя треугольные матрицы, k - номер текущей итерации, ω - итерационный параметр (при ω =1 метод верхней релаксации переходит в метод Зейделя). В данном методе результат постепенно приближается к искомым корням уравнения.

Вычислительная устойчивость метода Гаусс:

Метод Гаусса является вычислительно неустойчивым для плохо обусловленных матриц (то есть для матриц с большим числом обусловленности). Тем не менее примеры, в которых число обусловленности велико, встречаются крайне редко, поэтому метод широко употребляется. Уменьшить вычислительную ошибку можно с помощью метода Гаусса с выбором главного элемента, который является условно устойчивым.

Скорость сходимости метода релаксации:

Необходимое условие сходимости метода — $0 < \omega < 2$. Скорость сходимости метода определяется выбором параметра ω. При оптимальном значении можно уменьшить число итераций с $O(n^2)$до $O(n)$.

Описание программы

На вход программе подается число *var* (1 или 2). Если var = 2, то программа будет использовать для вычислений матрицу из приложения 2 (п. 1-6). Если *var* = 1, то программа будет считывать матрицу со стандартного потока ввода, предварительно считав размер матрицы N.

Программа ориентирована на работу с пользователем (то есть с ручным вводом), однако ввод в нее данных из файла можно организовать с помощью стандартной конструкции перенаправления ввода в bash, а именно команды < .

Для каждой системы уравнений программа выводит:

- определитель исходной матрицы,
- обратную матрицу,
- решение системы, полученное методом Гаусса и методом Гаусса с выбором главного элемента,
- количество операций сложения, умножения и деления, потребовавшихся для этого метода.

В методе верхней релаксации для каждой системы уравнений изучаем скорость сходимости итераций к точному решению задачи в зависимости от итерационного параметра ω. Считаем количество итераций для 0.01 <= ω <= 2, с шагом 0.1. Программа выводит для каждого ω число итераций, которое потребовалось. Затем программа выводит ω = fast_omega, при котором метод верхней релаксации сошёлся наиболее быстро, а также выводит решение, полученное с помощью этого метода и количество произведённых операций сложения, умножения и деления.

Текст программы

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

enum { INPUT_FILE = 1, INPUT_FORMULAS = 2 };
typedef double MX_TYPE;

int n = 25, m = 10;
long long int addG = 0, multG = 0, divG = 0;
long long int iter_cnt = 0, min_iter_cnt = -1, max_iter_cnt = 20000;
double fast_omega = 0.0;
long long int addR = 0, multR = 0, divR = 0;
long long int addRfast = 0, multRfast = 0, divRfast = 0;

// выделяем память под матрицу размера row_cnt на col_cnt
MX_TYPE**
init(int row_cnt, int col_cnt)
{
    MX_TYPE* *matrix = (MX_TYPE**) malloc(row_cnt * sizeof(MX_TYPE*));

    for (int i = 0; i < row_cnt; ++i) {
        matrix[i] = (MX_TYPE*) malloc(col_cnt * sizeof(MX_TYPE));
    }

    return matrix;
}
```

```
// освобождаем память под матрицу
void
free_matrix_space(MX_TYPE* *matrix, int N)
{
    for (int i = 0; i < N; ++i) {
        free(matrix[i]);
    }

    if (N > 1) {
        free(matrix);
    }
}

//печать матрицы
void
print_matrix(MX_TYPE* *matrix, int row_cnt, int col_cnt)
{
    for (int i = 0; i < row_cnt; ++i) {
        for (int j = 0; j < col_cnt; ++j) {
            printf("%.10g ", matrix[i][j]);
        }

        printf("\n");
    }
}

// копировать числовые значения матрицы old_matrix в матрицу new_matrix
void
copy_matrix_values(MX_TYPE* *old_matrix, int row_cnt, int col_cnt, MX_TYPE*
*new_matrix)
{
    for(int i = 0; i < row_cnt; ++i) {
        for (int j = 0; j < col_cnt; ++j) {
            new_matrix[i][j] = old_matrix[i][j];
        }
    }
}

// сделать матрицу единичной
void
init_identity_matrix(MX_TYPE* *matrix, int mxsize)
{
    for (int i = 0; i < mxsize; ++i) {
        for (int j = 0; j < mxsize; ++j) {
            if (j != i) {
                matrix[i][j] = 0;
            } else {
                matrix[i][j] = 1;
            }
        }
    }
}

// ищем первый после строки с номером cur_row ненулевой элемент и возвращаем
номер строки
MX_TYPE
get_leading_matrix_el_idx(MX_TYPE* *matrix, int mxsize, int cur_row)
{
    int index = cur_row;

    for (int k = cur_row + 1; k < mxsize; ++k) {
        if (matrix[k][cur_row] != 0) {
            index = k;
            break;
```

```
        }
    }

    return index;
}

// меняем строки с номерами row1 и row2 местами
void
swap_matrix_rows(MX_TYPE* *matrix, int col_cnt, int row1, int row2)
{
    for (int k = 0; k < col_cnt; ++k) {
        MX_TYPE temp = matrix[row1][k];
        matrix[row1][k] = matrix[row2][k];
        matrix[row2][k] = temp;
    }
}

// ищем ведущий элемент и меняем строки местами
int
Gauss_transform_matrix(MX_TYPE* *matrix, int mxsize, int cur_row)
{
    // ищем первый ненулевой элемент
    int lead_idx = get_leading_matrix_el_idx(matrix, mxsize, cur_row);

    //меняем строки с номерами cur_row и lead_idx местами
    swap_matrix_rows(matrix, mxsize, cur_row, lead_idx);

    return lead_idx;
}

//нахождение обратной матрицы методом Гаусса
void
inverse_matrix(MX_TYPE* *matr, int mxsize, MX_TYPE* *invmatrix)
{
    MX_TYPE* *matrix = init(mxsize, mxsize);
    copy_matrix_values(matr, mxsize, mxsize, matrix);

    // сделаем обратную матрицу единичной
    init_identity_matrix(invmatrix, mxsize);

    // прямой ход методом Гаусса
    for (int i = 0; i < mxsize - 1; ++i) {
        for (int j = i + 1; j < mxsize; ++j) {
            //если ведущий элемент равен 0
            if (matrix[i][i] == 0) {
                // ищем ненулевой элемент
                int lead_idx = get_leading_matrix_el_idx(matrix, mxsize, 0);

                //меняем строки местами
                swap_matrix_rows(matrix, mxsize, i, lead_idx);
                swap_matrix_rows(invmatrix, mxsize, i, lead_idx);
            }

            //вычисление коэффициента
            double coef = matrix[j][i] / matrix[i][i];

          //вычитание из j-той строчки i-той строки, умноженной на коэффициент
            for (int k = 0; k < mxsize; ++k) {
                matrix[j][k] -= coef * matrix[i][k];
                invmatrix[j][k] -= coef * invmatrix[i][k];
            }
        }
    }

    //обратный ход вычисления элементов обратной матрицы
    for (int i = mxsize - 1; i >= 0; --i) {
```

```
        for (int j = i; j > 0; --j) {
            double coef = matrix[j - 1][i] / matrix[i][i];

            for (int k = mxsize - 1; k >= 0; --k) {
                invmatrix[j - 1][k] -= invmatrix[i][k] * coef;
            }
        }
    }

    for (int i = 0; i < mxsize; i++) {
        for (int j = 0; j < mxsize; j++) {
            invmatrix[i][j] /= matrix[i][i];
        }
    }
}

//прямой ход метода Гаусса
double
Gauss_forward_elimination(MX_TYPE* *matrix, MX_TYPE mxsize, MX_TYPE *f)
{
    double matrix_det = 1;
    int sgn = 1; // sign of the determinant

    for (int i = 0; i < mxsize - 1; ++i) { // current row
        for (int j = i + 1; j < mxsize; ++j) { // next rows
            //если ведущий элемент равен 0
            if (matrix[i][i] == 0) {
              int leading_el_row = Gauss_transform_matrix(matrix, mxsize, i);

                if (matrix[leading_el_row][i] != 0.0) {
                    MX_TYPE temp = f[i];
                    f[i] = f[leading_el_row];
                    f[leading_el_row] = temp;

                    sgn = -sgn;
                } else {
                    break;
                }
            }

            //вычисление коэффициента
            double coef = matrix[j][i] / matrix[i][i];
            ++divG;

         //вычитание из j-той строчки i-той строки, умноженной на коэффициент
            for (int k = i; k < mxsize; ++k) {
                matrix[j][k] -= coef * matrix[i][k];

                ++addG;
                ++multG;
            }

            f[j] -= f[i] * coef;

            ++addG;
            ++multG;
        }
    }

    for (int i = 0; i < mxsize; ++i) {
        matrix_det *= matrix[i][i];
    }
    matrix_det *= sgn;

    return matrix_det;
}
```

```
// обратный ход метода Гаусса
void
Gauss_back_substitution(MX_TYPE*  *matrix,  int  mxsize,  MX_TYPE  *f,  double
*res)
{
    //обратный ход метода Гаусса
    if (matrix[mxsize - 1][mxsize - 1] == 0) {
        res[mxsize - 1] = 0;
    } else {
        res[mxsize - 1] = f[mxsize - 1] / matrix[mxsize - 1][mxsize - 1];
    }

    ++divG;

    for (int i = mxsize - 2; i >= 0; --i) {
        MX_TYPE temp = f[i];

        for (int j = mxsize - 1; j > i; --j) {
            temp -= matrix[i][j] * res[j];

            ++addG;
            ++multG;
        }

        if (temp == 0) {
            res[i] = 0.0;
        } else {
            res[i] = (double) temp / matrix[i][i];
        }

        ++divG;
    }
}

//Метод Гаусса
double
Gauss(MX_TYPE* *matrix, int mxsize, MX_TYPE *f, double *res)
{
    MX_TYPE* *matrix_copy = init(mxsize, mxsize);
    copy_matrix_values(matrix, mxsize, mxsize, matrix_copy);

    MX_TYPE* *f_copy = init(1, mxsize);
    copy_matrix_values(&f, 1, mxsize, f_copy);

     double  matrix_det  =  Gauss_forward_elimination(matrix_copy,  mxsize,
*f_copy);
    if (matrix_det != 0) {
        Gauss_back_substitution(matrix_copy, mxsize, *f_copy, res);
    }

    free_matrix_space(matrix_copy, mxsize);
    free_matrix_space(f_copy, 1);

    return matrix_det;
}

// поиск максимального по модулю элемента в строке с номером cur_row и
возврат номера столбца
int
get_idx_row_max(MX_TYPE* *matrix, int col_cnt, int cur_row)
{
    MX_TYPE max = fabs(matrix[cur_row][cur_row]);
```

```
    int col_idx = cur_row;

    for (int k = cur_row + 1; k < col_cnt; ++k) {
        if (fabs(matrix[cur_row][k]) > max) {
            col_idx = k;
            max = fabs(matrix[cur_row][k]);
        }
    }

    return col_idx;
}

// переставить местами столбцы с номерами col1 и col2
void
swap_matrix_columns(MX_TYPE* *matrix, int row_cnt, int col1, int col2)
{
    for (int k = 0; k < row_cnt; ++k) {
        MX_TYPE temp = matrix[k][col1];
        matrix[k][col1] = matrix[k][col2];
        matrix[k][col2] = temp;
    }
}

//Метод Гаусса с выбором главного элемента
void
Gauss_select(MX_TYPE* *matrix, int mxsize, MX_TYPE *f, double *res)
{
    MX_TYPE* *matrix_copy = init(mxsize, mxsize);
    copy_matrix_values(matrix, mxsize, mxsize, matrix_copy);

    MX_TYPE* *f_copy = init(1, mxsize);
    copy_matrix_values(&f, 1, mxsize, f_copy);

    double temp;

    int *res_num = malloc(mxsize * sizeof(int));
    for (int i = 0; i < mxsize; ++i) {
        res_num[i] = i;
    }

    //прямой ход метода Гаусса
    for (int i = 0; i < mxsize - 1; ++i) {
        //поиск максимального по модулю элемента в строке
        int index = get_idx_row_max(matrix_copy, mxsize, i);

        //меняем столбцы с номерами i и index местами
        if (i != index) {
            swap_matrix_columns(matrix_copy, mxsize, i, index);
        }

         int buf = res_num[i];
         res_num[i] = res_num[index];
         res_num[index] = buf;

         for (int j = i + 1; j < mxsize; j++) {
            //вычисление коэффициента
            temp = matrix_copy[j][i] / matrix_copy[i][i];

          //вычитание из j-той строки i-той строки, умноженной на коэффициент
            for (int k = i; k < mxsize; k++) {
                matrix_copy[j][k] -= temp * matrix_copy[i][k];
            }

            (*f_copy)[j] -= temp * (*f_copy)[i];
        }
    }
```

```
    //обратный ход метода Гаусса
    Gauss_back_substitution(matrix_copy, mxsize, *f_copy, res);

    for (int i = 0; i < mxsize; ++i) {
        if (res_num[i] != i) {
            temp = res[i];
            res[i] = res[res_num[i]];
            res[res_num[i]] = temp;

            int buf = res_num[i];
            res_num[i] = res_num[buf];
            res_num[buf] = buf;
        }
    }

    free_matrix_space(matrix_copy, mxsize);
    free_matrix_space(f_copy, 1);
}

//создание матрицы по примеру из приложения 2
void
create_matrix(MX_TYPE* *matrix, MX_TYPE *f)
{
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i != j) {
                matrix[i][j] = (i + j) / (m + n);
            } else {
                matrix[i][j] = n + pow(m, 2) + j/m + i/n;
            }
        }

        f[i] = i * i - n;
    }
}

// Вычисление нормы разности 2х векторов
long double
vector_diff(double *prev_x, double *curr_x, int mxsize)
{
    long double diff = 0.0;

    for (int i = 0; i < mxsize; ++i) {
        diff += pow((curr_x[i] - prev_x[i]), 2);
    }

    return sqrt(diff);
}

// вычисление очередной итерации в методе верхней релаксации
long double
SOR_next_iter(MX_TYPE*  *matrix,  int  mxsize,  MX_TYPE  *f,  double  *prev_x,
double *curr_x, double omega)
{
    double sum;
    int i, j;

    for (i = 0; i < mxsize; ++i) {
        sum = 0.0;

        for (j = 0; j < i; ++j) {
            sum += -(matrix[i][j] * curr_x[j]);

            ++addR;
            ++multR;
```

```
        }

        for (j = i; j < mxsize; ++j) {
            sum += -(matrix[i][j] * prev_x[j]);
            ++addR;
            ++multR;
        }

        sum += f[i];
        sum *= omega;
        sum /= matrix[i][i];
        curr_x[i] = prev_x[i] + sum;

        addR += 2;
        multR += 2;
        ++divR;
    }

    return vector_diff(prev_x, curr_x, mxsize);
}

// x имеет конечную норму?
int
is_finite(double *x, int x_size)
{
    for (int i = 0; i < x_size; ++i) {
        if (!isfinite(x[i])) {
            // x[i] - бесконечность или не число
            return 0;
        }
    }

    return 1;
}

// метод верхней релаксации
int
SOR(MX_TYPE* *matrix, int mxsize, MX_TYPE *f, double *fast_x, double eps)
{
    int SOR_converges = 0;
    int converge; // флаг для сходимости

    double omega;
   double conv_max_omega = 2.0; // максимальное значение омега для сходимости
метода
    double start_omega = 0.1;
    double omega_step = 0.1;

    double *prev_x = malloc(mxsize * sizeof(double));
    double *curr_x = malloc(mxsize * sizeof(double));
    double *tmp = NULL;

      for (omega = start_omega; omega <= conv_max_omega; omega +=omega_step)
{ //смотрим для 0 < omega <= 2 сходимость
        for (int i = 0; i < mxsize; ++i) {
             prev_x[i] = 0.0; // принимаем за начальное приближение нулевой
вектор
        }

        addR = 0;
        multR = 0;
        divR = 0;

        converge = 1;
        iter_cnt = 0;
```

```
        while (SOR_next_iter(matrix, mxsize, f, prev_x, curr_x, omega) > eps)
{
            ++iter_cnt;
            if (iter_cnt > max_iter_cnt) {
                converge = 0;
                break;
            }

            tmp = prev_x;
            prev_x = curr_x;
            curr_x = tmp;
        }

        if (converge) {
            converge = is_finite(curr_x, mxsize);
        }

        if (converge) {
            printf("omega = %f %lld iterations\n", omega, iter_cnt);
            if ((omega < start_omega * 1.1) || (iter_cnt < min_iter_cnt)) {
                min_iter_cnt = iter_cnt;
                SOR_converges = 1;

                for (int i = 0; i < mxsize; ++i) {
                    fast_x[i] = curr_x[i];
                }

                addRfast = addR;
                multRfast = multR;
                divRfast = divR;

                fast_omega = omega;
            }
        } else {
            printf("omega = %f method diverges\n", omega);
        }
    }

    return SOR_converges;
}

int
main(int argc, char *argv[])
{
    int i, j;

    int var;
    do {
        printf("How do you want to input data?\n");
        printf("Using input file - input %d\n", INPUT_FILE);
        printf("Using formulas - input %d\n", INPUT_FORMULAS);
        scanf("%d", &var);
    } while (var != INPUT_FILE && var != INPUT_FORMULAS);

    int N; // размер матрицы
    if (var == INPUT_FILE) {
        do {
            printf("Input matix size:\n");
            scanf("%d", &N);
        } while (N < 1);
    } else {
        N = n;
    }

    MX_TYPE* *matrix = init(N, N); // исходная матрица
    MX_TYPE* *invmatrix = init(N, N); // обратная матрица
```

```
    MX_TYPE *f = malloc(N * sizeof(MX_TYPE)); // вектор значений
    MX_TYPE *res = malloc(N * sizeof(MX_TYPE)); // вектор решений

    if (var == INPUT_FILE) {
        printf("INPUT FORMAT:\n");
        printf("a11 a12 ... a1n f1\na21 a22 ... a2n f2\n... ... ... ... ...
\nan1 an2 ... ann fn\n\n");
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                scanf("%lf", &matrix[i][j]);
            }

            scanf("%lf", &f[i]);
        }
    } else {
        if (var == INPUT_FORMULAS) {
            create_matrix(matrix, f);
        }
    }

    double matrix_det = Gauss(matrix, N, f, res);
    printf("Determinant of matrix: %.10g\n», matrix_det);

    if (matrix_det != 0.0) {
         inverse_matrix(matrix, N, invmatrix); // вычисляем обратную матрицу
         printf("Inverse matrix:\n");
         print_matrix(invmatrix, N, N);
         free_matrix_space(invmatrix, N);

         printf("Roots of SLAE — Gauss' method:\n"); // решаем систему методом
Гаусса
        for (int i = 0; i < N; ++i) {
            printf("x%d = %f\n", i + 1, res[i]);
        }

        long long int add_cnt = addG, mult_cnt = multG, div_cnt = divG;

        Gauss_select(matrix, N, f, res);
        printf("Roots of SLAE — Gauss' method with selection:\n"); // решаем
систему методом Гаусса с выбором главного элемента
        for (int i = 0; i < N; ++i) {
            printf("x%d = %f\n", i + 1, res[i]);
        }

        printf("Operations needed for Gauss' method and Gauss' method with
selection:\n");
          printf(" addition %lld\n multiplication %lld\n division %lld\n",
add_cnt, mult_cnt, div_cnt);

        // Метод верхней релаксации
        double eps;
        printf("Input epsilon:\n");
        do {
            scanf("%lf", &eps);
        } while (eps <= 0.0);

        int converge = SOR(matrix, N, f, res, eps);
        if (converge) {
            printf("method converges the most quickly with omega = %f\n",
fast_omega);
            printf("Roots of SLAE — successive over-relaxation method:\n");
            for (i = 0; i < N; ++i) {
                printf("x%d = %f\n", i + 1, res[i]);
            }
            printf("Operations needed for successive over-relaxation method:
\n");
```

```
            printf("%lld iterations\n\%lld addition\n%lld multiplication\n%lld division\n", min_iter_cnt, addRfast, multRfast, divRfast);
        } else {
            printf("%s\n", "successive over-relaxation method diverges");
        }
    } else {
        printf("The matrix is degenerate! There is no the one solution!\n");
    }

    free_matrix_space(matrix, N);
    free_matrix_space(&f, 1);
    free_matrix_space(&res, 1);

    return 0;
}
```

Тестирование

Входные данные подавались в файле input.txt. Вывод осуществлялся в файл output.txt.

Все решения были проверены с помощью ресурса http://matrix.reshish.ru/.

Тест 1

Ввод:

```
1
4
2 3 11 5 2
1 1 5 2 1
2 1 3 2 -3
1 1 3 4 -3
0.0000001
```

Вывод:

```
Determinant of matrix: 14
Inverse matrix:
-0.2857142857 0.2857142857 0.7142857143 -0.1428571429
1.285714286 -2.785714286 0.2857142857 -0.3571428571
-0.1428571429 0.6428571429 -0.1428571429 -0.07142857143
-0.1428571429 0.1428571429 -0.1428571429 0.4285714286
Roots of SLAE — Gauss' method:
x1 = -2.000000
x2 = 0.000000
x3 = 1.000000
x4 = -1.000000
Roots of SLAE — Gauss' method with selection:
x1 = -2.000000
x2 = -0.000000
x3 = 1.000000
x4 = -1.000000
Operations needed for Gauss' method and Gauss' method with selection:
 addition 32
 multiplication 32
 division 10
Input epsilon:
omega = 0.100000 method diverges
omega = 0.200000 method diverges
omega = 0.300000 method diverges
omega = 0.400000 method diverges
omega = 0.500000 method diverges
omega = 0.600000 method diverges
omega = 0.700000 method diverges
omega = 0.800000 method diverges
omega = 0.900000 method diverges
omega = 1.000000 method diverges
omega = 1.100000 method diverges
omega = 1.200000 method diverges
omega = 1.300000 method diverges
omega = 1.400000 method diverges
omega = 1.500000 method diverges
omega = 1.600000 method diverges
omega = 1.700000 method diverges
```

```
omega = 1.800000 method diverges
omega = 1.900000 method diverges
successive over-relaxation method diverges
```

Тест 2

Ввод:

```
1
4
2 -1 1 2 2
6 -3 2 4 3
6 -3 4 8 9
4 -2 1 1 1
0.0000001
```

Вывод:

```
Determinant of matrix: 0
The matrix is degenerate! There is no the one solution!
```

Тест 3

Ввод:

```
1
4
1 1 -3 1 -1
2 1 -2 0 1
1 1 1 0 3
1 2 -3 -7 1
0.0000001
```

Вывод:

```
Determinant of matrix: 36
Inverse matrix:
-0.5833333333 0.9166666667 -0.1666666667 -0.08333333333
0.7777777778 -0.8888888889 0.8888888889 0.1111111111
-0.1944444444 -0.02777777778 0.2777777778 -0.02777777778
0.2222222222 -0.1111111111 0.1111111111 -0.1111111111
Roots of SLAE — Gauss' method:
x1 = 0.916667
x2 = 1.111111
x3 = 0.972222
x4 = -0.111111
Roots of SLAE — Gauss' method with selection:
x1 = 0.916667
x2 = 1.111111
x3 = 0.972222
x4 = -0.111111
Operations needed for Gauss' method and Gauss' method with selection:
 addition 32
 multiplication 32
 division 10
Input epsilon:
omega = 0.100000 method diverges
omega = 0.200000 method diverges
omega = 0.300000 method diverges
omega = 0.400000 method diverges
omega = 0.500000 method diverges
omega = 0.600000 method diverges
omega = 0.700000 method diverges
omega = 0.800000 method diverges
omega = 0.900000 method diverges
omega = 1.000000 method diverges
omega = 1.100000 method diverges
omega = 1.200000 method diverges
omega = 1.300000 method diverges
omega = 1.400000 method diverges
omega = 1.500000 method diverges
omega = 1.600000 method diverges
omega = 1.700000 method diverges
omega = 1.800000 method diverges
omega = 1.900000 method diverges
successive over-relaxation method diverges
```

Тест 4

Матрица из приложения 2 (п. 1-6)

Ввод:

```
2
0.0000001
```

Вывод:

```
Determinant of matrix: 3.094086222e+52
Inverse matrix:
0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0.008 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0.007936507937 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0.007937004292 4.927735674e-07 4.891606711e-07
4.855169634e-07 4.81842674e-07 4.78108397e-07 4.74343764e-07 4.74343764e-07
4.705188968e-07 4.629604716e-07 4.591071624e-07 4.552249348e-07 4.513140335e-07
-6.254078352e-05
0 0 0 0 0 0 0 0 0 0 0 4.927735674e-07 0.007937493453 9.782905308e-07
9.71003345e-07 9.636549975e-07 9.561866788e-07 9.486576499e-07 9.486576499e-07
9.410081564e-07 9.258917822e-07 9.181854064e-07 9.104211959e-07
-6.208555859e-05 -6.208946949e-05
0 0 0 0 0 0 0 0 0 0 0 4.891606711e-07 9.782905308e-07 0.007937975295
1.456428565e-06 1.445406622e-06 1.434204732e-06 1.422911782e-06 1.422911782e-06
1.411438144e-06 1.388764773e-06 1.377205816e-06 -6.162259812e-05
-6.163036233e-05 -6.163424455e-05
0 0 0 0 0 0 0 0 0 0 0 4.855169634e-07 9.71003345e-07 1.456428565e-06
0.007938449699 1.927067203e-06 1.912132448e-06 1.897076287e-06 1.897076287e-06
1.881779227e-06 1.851550287e-06 -6.115201875e-05 -6.116357771e-05
-6.117128408e-05 -6.117513739e-05
0 0 0 0 0 0 0 0 0 0 0 4.81842674e-07 9.636549975e-07 1.445406622e-06
1.927067203e-06 0.007938916543 2.389939722e-06 2.371121298e-06 2.371121298e-06
2.352001781e-06 -6.067393905e-05 -6.068923323e-05 -6.070070472e-05
-6.070835277e-05 -6.071217692e-05
0 0 0 0 0 0 0 0 0 0 0 4.78108397e-07 9.561866788e-07 1.434204732e-06
1.912132448e-06 2.389939722e-06 0.007939379354 2.848808123e-06 2.848808123e-06
-6.066622671e-05 -6.02037165e-05 -6.021889215e-05 -6.023027473e-05
-6.023786351e-05 -6.024165802e-05
0 0 0 0 0 0 0 0 0 0 0 4.74343764e-07 9.486576499e-07 1.422911782e-06
1.897076287e-06 2.371121298e-06 2.848808123e-06 0.007939834251 -6.016574942e-05
-6.018853989e-05 -5.972967148e-05 -5.974472765e-05 -5.97560206e-05
-5.976354963e-05 -5.976731426e-05
0 0 0 0 0 0 0 0 0 0 0 4.74343764e-07 9.486576499e-07 1.422911782e-06
1.897076287e-06 2.371121298e-06 2.848808123e-06 -6.016574942e-05 0.007939834251
-6.018853989e-05 -5.972967148e-05 -5.974472765e-05 -5.97560206e-05
-5.976354963e-05 -5.976731426e-05
0 0 0 0 0 0 0 0 0 0 0 4.705188968e-07 9.410081564e-07 1.411438144e-06
1.881779227e-06 2.352001781e-06 -6.066622671e-05 -6.018853989e-05
-6.018853989e-05 0.00794029679 -5.924804175e-05 -5.926297651e-05
-5.92741784e-05 -5.928164672e-05 -5.928538099e-05
0 0 0 0 0 0 0 0 0 0 0 4.629604716e-07 9.258917822e-07 1.388764773e-06
1.851550287e-06 -6.067393905e-05 -6.02037165e-05 -5.972967148e-05
-5.972967148e-05 -5.924804175e-05 0.007878211657 -5.831097484e-05
-5.832199679e-05 -5.832934513e-05 -5.833301942e-05
0 0 0 0 0 0 0 0 0 0 0 4.591071624e-07 9.181854064e-07 1.377205816e-06
-6.115201875e-05 -6.068923323e-05 -6.021889215e-05 -5.974472765e-05
-5.974472765e-05 -5.926297651e-05 -5.831097484e-05 0.007878682295
-5.783657157e-05 -5.784385875e-05 -5.784750246e-05
0 0 0 0 0 0 0 0 0 0 0 4.552249348e-07 9.104211959e-07 -6.162259812e-05
-6.116357771e-05 -6.070070472e-05 -6.023027473e-05 -5.97560206e-05
-5.97560206e-05 -5.92741784e-05 -5.832199679e-05 -5.783657157e-05
0.007879160433 -5.735472889e-05 -5.735834179e-05
0 0 0 0 0 0 0 0 0 0 0 4.513140335e-07 -6.208555859e-05 -6.163036233e-05
-6.117128408e-05 -6.070835277e-05 -6.023786351e-05 -5.976354963e-05
```

```
-5.976354963e-05 -5.928164672e-05 -5.832934513e-05 -5.784385875e-05
-5.735472889e-05 0.00787964595 -5.686556822e-05
0 0 0 0 0 0 0 0 0 0 0 -6.254078352e-05 -6.208946949e-05 -6.163424455e-05
-6.117513739e-05 -6.071217692e-05 -6.024165802e-05 -5.976731426e-05
-5.976731426e-05 -5.928538099e-05 -5.833301942e-05 -5.784750246e-05
-5.735834179e-05 -5.686556822e-05 0.007880138724
Roots of SLAE — Gauss' method:
x1 = -0.200000
x2 = -0.192000
x3 = -0.168000
x4 = -0.128000
x5 = -0.072000
x6 = 0.000000
x7 = 0.088000
x8 = 0.192000
x9 = 0.312000
x10 = 0.448000
x11 = 0.595238
x12 = 0.729138
x13 = 0.881824
x14 = 1.053163
x15 = 1.243017
x16 = 1.451249
x17 = 1.677562
x18 = 1.921990
x19 = 2.201990
x20 = 2.484570
x21 = 2.762857
x22 = 3.078389
x23 = 3.411300
x24 = 3.761444
x25 = 4.128673
Roots of SLAE — Gauss' method with selection:
x1 = -0.200000
x2 = -0.192000
x3 = -0.168000
x4 = -0.128000
x5 = -0.072000
x6 = 0.000000
x7 = 0.088000
x8 = 0.192000
x9 = 0.312000
x10 = 0.448000
x11 = 0.595238
x12 = 0.729138
x13 = 0.881824
x14 = 1.053163
x15 = 1.243017
x16 = 1.451249
x17 = 1.677562
x18 = 1.921990
x19 = 2.201990
x20 = 2.484570
x21 = 2.762857
x22 = 3.078389
x23 = 3.411300
x24 = 3.761444
x25 = 4.128673
Operations needed for Gauss' method and Gauss' method with selection:
 addition 5800
 multiplication 5800
 division 325
Input epsilon:
omega = 0.100000 145 iterations
omega = 0.200000 72 iterations
omega = 0.300000 46 iterations
omega = 0.400000 33 iterations
omega = 0.500000 25 iterations
omega = 0.600000 19 iterations
omega = 0.700000 15 iterations
omega = 0.800000 12 iterations
omega = 0.900000 9 iterations
```

```
omega = 1.000000 5 iterations
omega = 1.100000 9 iterations
omega = 1.200000 13 iterations
omega = 1.300000 17 iterations
omega = 1.400000 21 iterations
omega = 1.500000 28 iterations
omega = 1.600000 38 iterations
omega = 1.700000 54 iterations
omega = 1.800000 87 iterations
omega = 1.900000 184 iterations
method converges the most quickly with omega = 1.000000
Roots of SLAE — successive over-relaxation method:
x1 = -0.200000
x2 = -0.192000
x3 = -0.168000
x4 = -0.128000
x5 = -0.072000
x6 = 0.000000
x7 = 0.088000
x8 = 0.192000
x9 = 0.312000
x10 = 0.448000
x11 = 0.595238
x12 = 0.729138
x13 = 0.881824
x14 = 1.053163
x15 = 1.243017
x16 = 1.451249
x17 = 1.677562
x18 = 1.921990
x19 = 2.201990
x20 = 2.484570
x21 = 2.762857
x22 = 3.078389
x23 = 3.411300
x24 = 3.761444
x25 = 4.128673
Operations needed for successive over-relaxation method:
5 iterations
4050 addition
4050 multiplication
150 division
```

Тест 5

Ввод:

```
1
4
2 -1 -6 3 -1
7 -4 2 -15 -32
1 -2 -4 9 5
1 -1 2 -6 -8
0.0000001
```

Вывод:

```
Determinant of matrix: -252
Inverse matrix:
-0.2142857143 0.380952381 -0.07142857143 -1.166666667
-0.07142857143 0.2380952381 -0.3571428571 -1.166666667
-0.2857142857 0.119047619 0.07142857143 -0.3333333333
-0.119047619 0.06349206349 0.07142857143 -0.2777777778
Roots of SLAE — Gauss' method:
x1 = -3.000000
x2 = 0.000000
x3 = -0.500000
x4 = 0.666667
Roots of SLAE — Gauss' method with selection:
x1 = -3.000000
x2 = -0.000000
x3 = -0.500000
x4 = 0.666667
Operations needed for Gauss' method and Gauss' method with selection:
```

```
 addition 32
 multiplication 32
 division 10
Input epsilon:
omega = 0.100000 method diverges
omega = 0.200000 method diverges
omega = 0.300000 method diverges
omega = 0.400000 method diverges
omega = 0.500000 method diverges
omega = 0.600000 method diverges
omega = 0.700000 method diverges
omega = 0.800000 method diverges
omega = 0.900000 method diverges
omega = 1.000000 method diverges
omega = 1.100000 method diverges
omega = 1.200000 method diverges
omega = 1.300000 method diverges
omega = 1.400000 method diverges
omega = 1.500000 method diverges
omega = 1.600000 method diverges
omega = 1.700000 method diverges
omega = 1.800000 method diverges
omega = 1.900000 method diverges
successive over-relaxation method diverges
```

Тест 6

Ввод:

```
1
4
1 -2 1 1 0
2 1 -1 -1 0
3 -1 -2 1 0
5 2 -1 9 -10
0.0000001
```

Вывод:

```
Determinant of matrix: -118
Inverse matrix:
0.2796610169 0.4237288136 -0.08474576271 0.02542372881
-0.1610169492 0.2711864407 -0.2542372881 0.07627118644
0.4661016949 0.3728813559 -0.4745762712 0.04237288136
-0.06779661017 -0.2542372881 0.05084745763 0.08474576271
Roots of SLAE — Gauss' method:
x1 = -0.254237
x2 = -0.762712
x3 = -0.423729
x4 = -0.847458
Roots of SLAE — Gauss' method with selection:
x1 = -0.254237
x2 = -0.762712
x3 = -0.423729
x4 = -0.847458
Operations needed for Gauss' method and Gauss' method with selection:
 addition 32
 multiplication 32
 division 10
Input epsilon:
omega = 0.100000 148 iterations
omega = 0.200000 77 iterations
omega = 0.300000 52 iterations
omega = 0.400000 38 iterations
omega = 0.500000 29 iterations
omega = 0.600000 method diverges
omega = 0.700000 method diverges
omega = 0.800000 method diverges
omega = 0.900000 method diverges
omega = 1.000000 method diverges
omega = 1.100000 method diverges
omega = 1.200000 method diverges
```

```
omega = 1.300000 method diverges
omega = 1.400000 method diverges
omega = 1.500000 method diverges
omega = 1.600000 method diverges
omega = 1.700000 method diverges
omega = 1.800000 method diverges
omega = 1.900000 method diverges
method converges the most quickly with omega = 0.500000
Roots of SLAE — successive over-relaxation method:
x1 = -0.254237
x2 = -0.762712
x3 = -0.423729
x4 = -0.847458
Operations needed for successive over-relaxation method:
29 iterations
720 addition
720 multiplication
120 division
```

Выводы

В ходе практической работы я рассмотрел методы решения систем линейных алгебраических уравнений методом Гаусса, методом Гаусса с выбором главного элемента и методом верхней релаксации. Данные методы были реализованы в виде программы на языке C, а также был написан алгоритм нахождения обратной матрицы и метод вычисления определителя матрицы.

Метод верхней релаксации не работает для некоторых матриц из примеров, так как они не являются положительно определёнными. Этот метод прост и удобен для вычислений (в отличие от метода Гаусса), так как он не требует никаких действий с исходной матрицей и не требует дополнительного резерва памяти, что существенно при решении больших систем.